클래스와 구조체

Classes & Structures

"*Classes* and *structures* are general-purpose, flexible constructs that become the building blocks of your program's code.
You define properties and methods to add functionality to your classes and structures by using exactly the same syntax as for constants, variables, and functions."

Classes & Structures

- 프로그램 코드 블럭의 기본 구조이다.
- 변수, 상수, 함수를 추가 할수 있다. (두 구조의 문법 같음)
- 단일 파일에 정의 되며 다른 코드에서 자동으로 사용 할수 있습니다.(접근 제한자에 따라 접근성은 차이가 있다. internal 기본 접근제한자)
- 초기 상태를 설정하기 위해 initializer가 만들어 지고, 사용자가 추가로 정의할 수 있다.
- 사용 시 인스턴스(instance)라고 불린다.
- 기본 구현된 내용에 기능을 더 추가해서 확장 할수 있다. (Extensions)
- 프로토콜을 상속받아 사용할수 있다. (Protocols)

기본 구조

```
class SomeClass {
    // class definition goes here
}
```

```
struct SomeStructure {
    // structure definition goes here
}
```

기본 구조

```
struct Resolution {
    var width = 0
    var height = 0
}
```

```
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

인스턴스

```
let someResolution = Resolution()
let someVideoMode = VideoMode()
```

Properties 접근

```
print("Width of Resolution is \(someResolution.width)”)
print("Width of VideoMode is \(someVideoMode.resolution.width)”)

someVideoMode.resolution.width = 1280
```

- 닷(.) 문법을 통해 접근

Initialization

Initialization is the process of preparing an instance of a class, structure, or enumeration for use.

초기화

- 인스턴스에 설정된 속성의 초기값을 설정과 초기화하는데 목적이 있다.
- 클래스 및 구조체는 인스턴스로 만들어 질때 프로퍼티는 적절한 초기값으로 모두 초기화 해야 한다.
- 모든 구조체는 자동으로 Memberwise Initializers가 만들어 진다.

base Initializers

```
struct Subject {
    var name:String
}

class Student {

    var subjects:[Subject] = []

    func addSubject(name:String) {

        let subject = Subject(name: name)
        subjects.append(subject)
    }
}

var wingMan:Person = Person()
```

Memberwise Initializers

Initializers

Custom Initializers

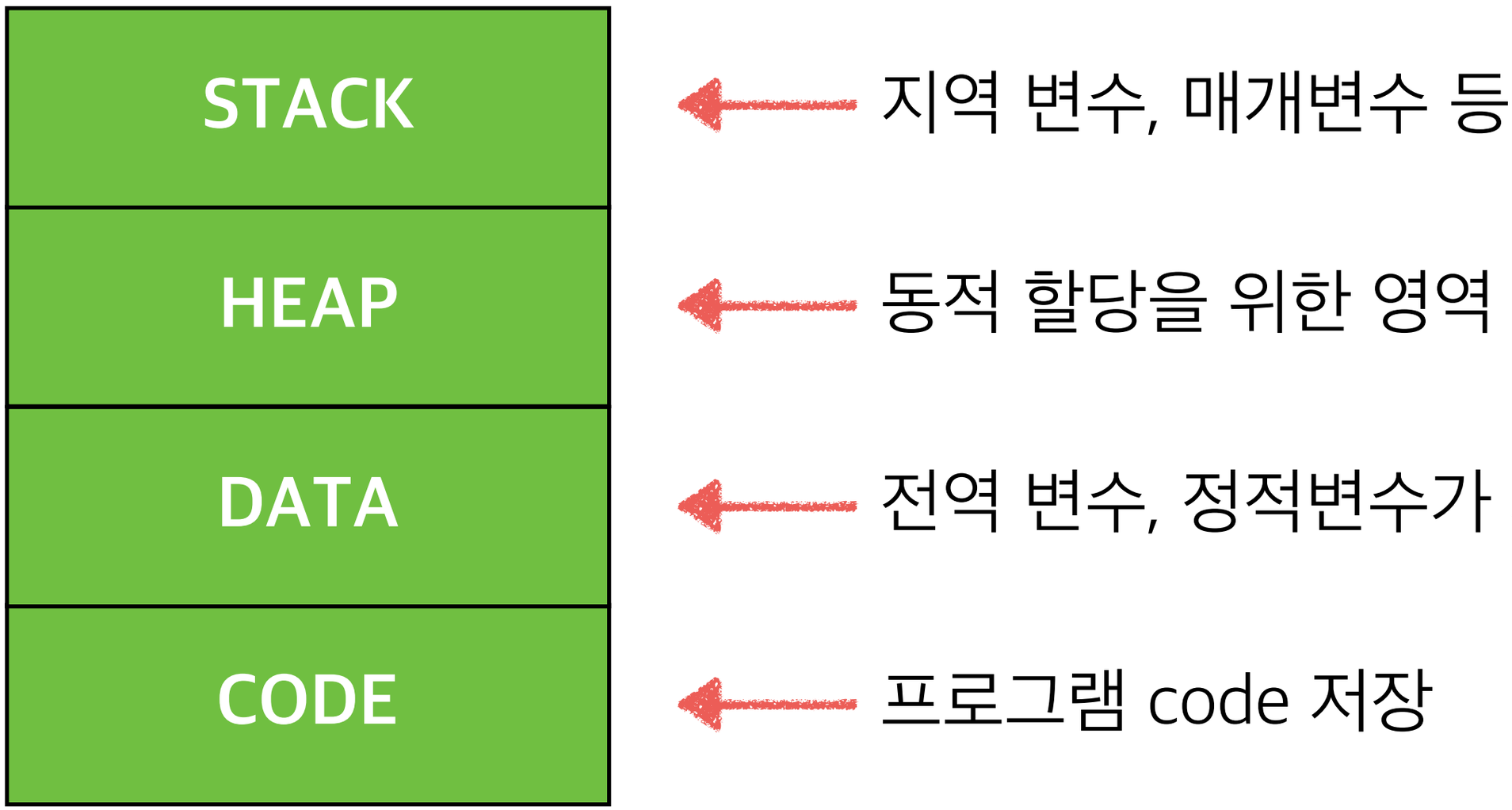
```
class Student {
    var subjects:[Subject] = []

    func addSubject(name:String) {
        var sub1:Subject = Subject(gender:true)
        sub1.name = "joo"
        sub1.age = 30
        subjects.append(sub1)
    }
}

struct Subject {
    var name:String?
    var age:Int?
    var gender:Bool

    init(gender:Bool) {
        self.gender = gender
    }
}
```

Classes VS Structures

- Class는 참조 타입이며, Structure는 값 타입이다.
- Class는 상속을 통해 부모클래스의 특성을 상속받을수 있다.
- Class는 Type Casting을 사용할수 있다.(Structure 불가)
- Structure의 프로퍼티는 instance가 var를 통해서 만들어야 수정 가능하다.
- Class는 Reference Counting을 통해 인스턴스의 해제를 계산합니다.
- Calss는 deinitializer를 사용할수 있습니다.

값타입 vs 참조타입

Memory구조

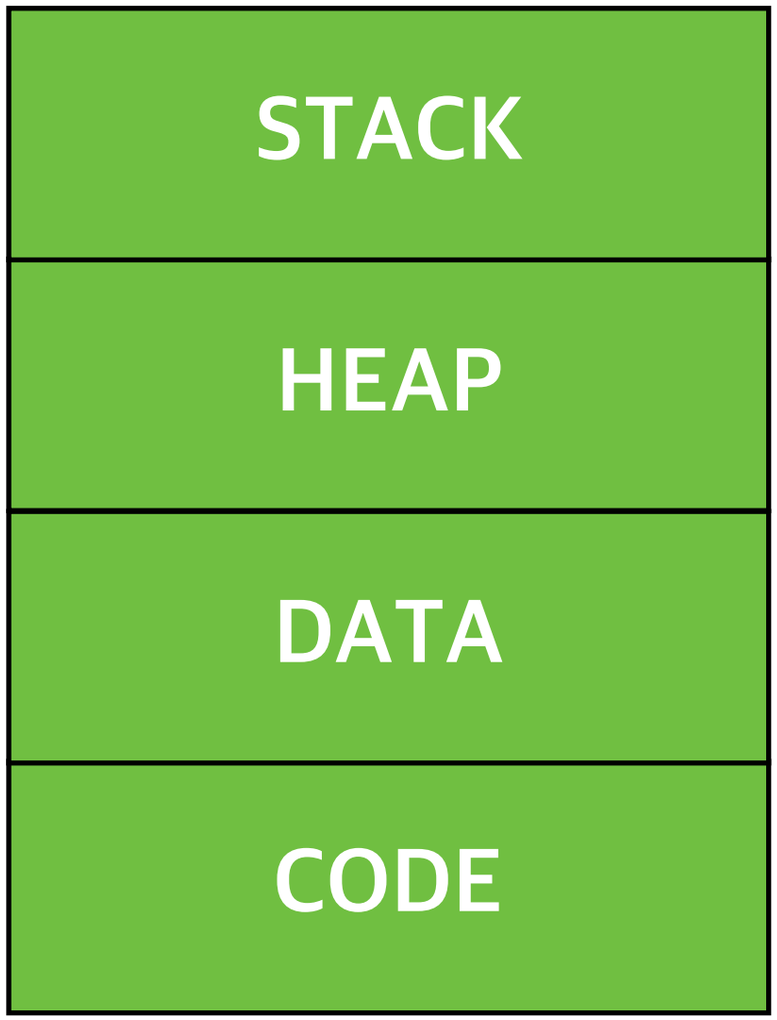


Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
var num:Int = 4
var num2:Int = 5
```

Memory구조 파악하기



a = 4, b = 5

프로그램 code 저장

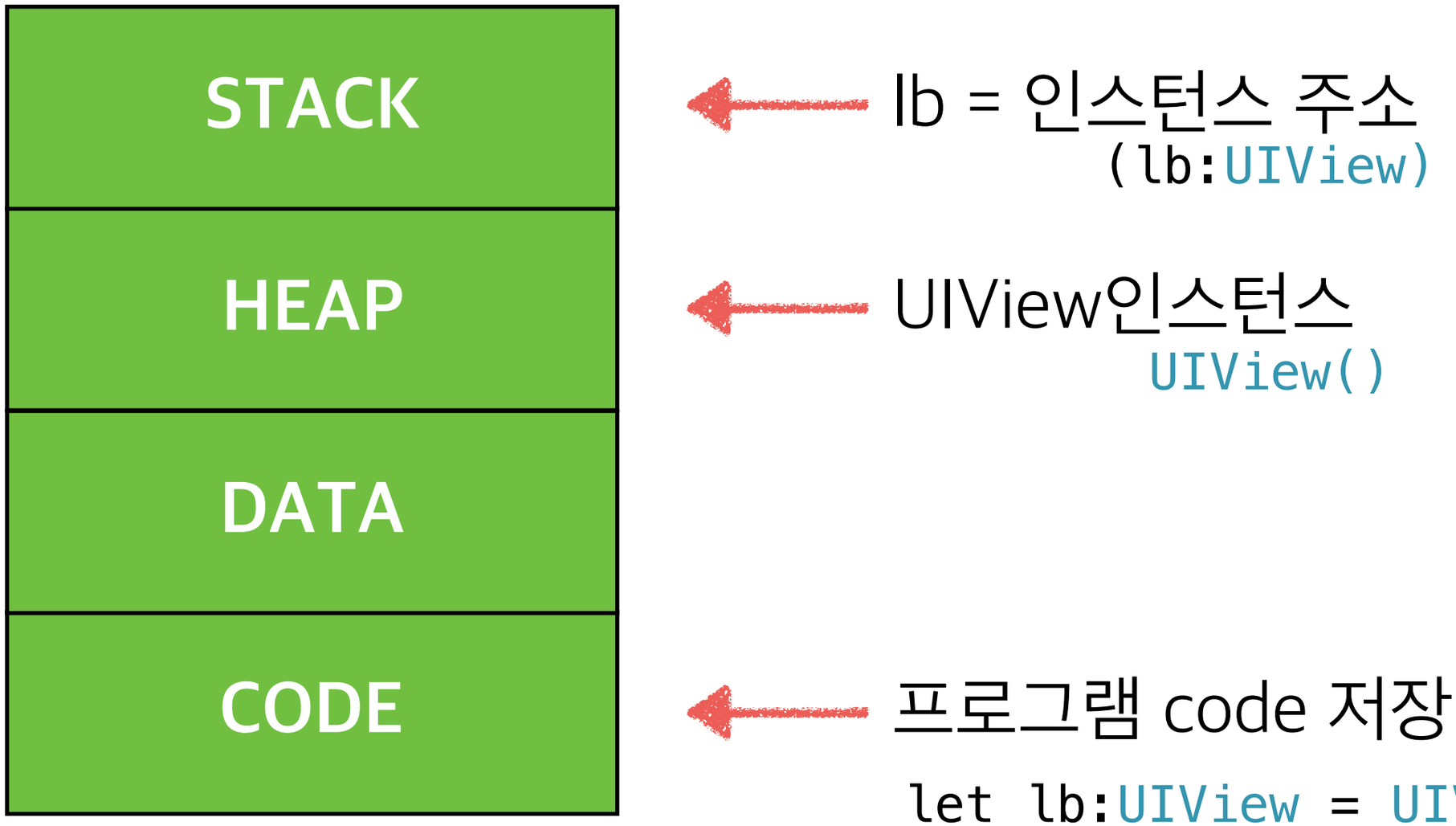
```
var num:Int = 4;
var num2:Int = 5;
```

Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
let lb:UIView = UIView()
```

Memory구조 파악하기

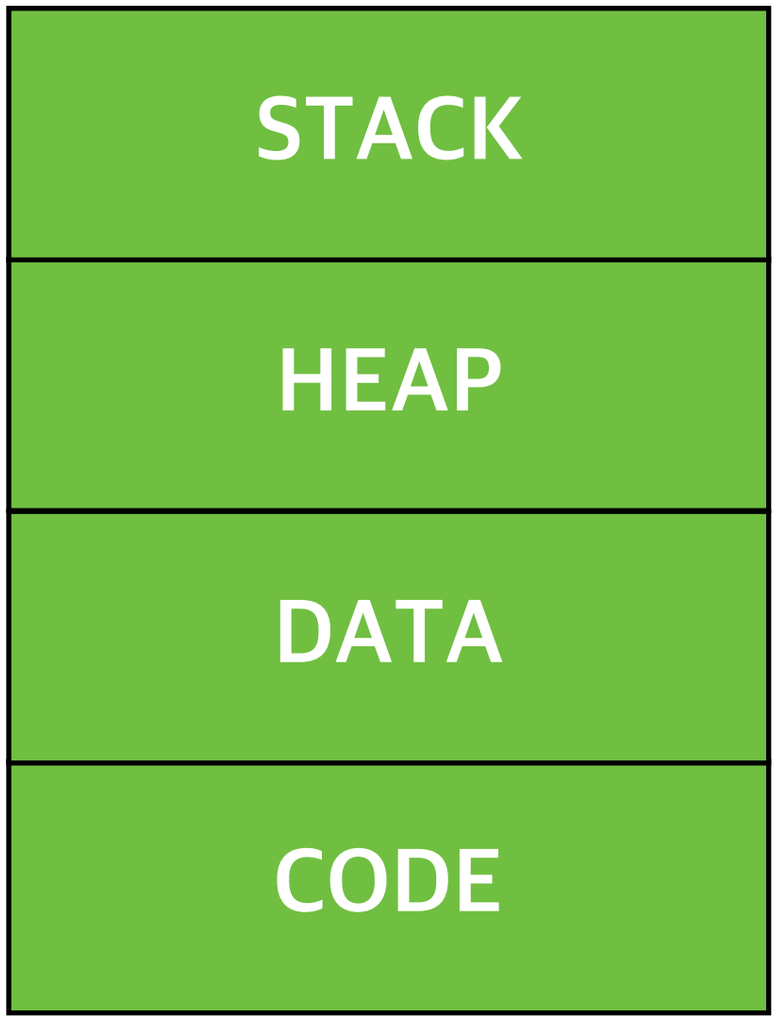


Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
static let number:Int = 5
```

Memory구조 파악하기



number = 5

프로그램 code 저장

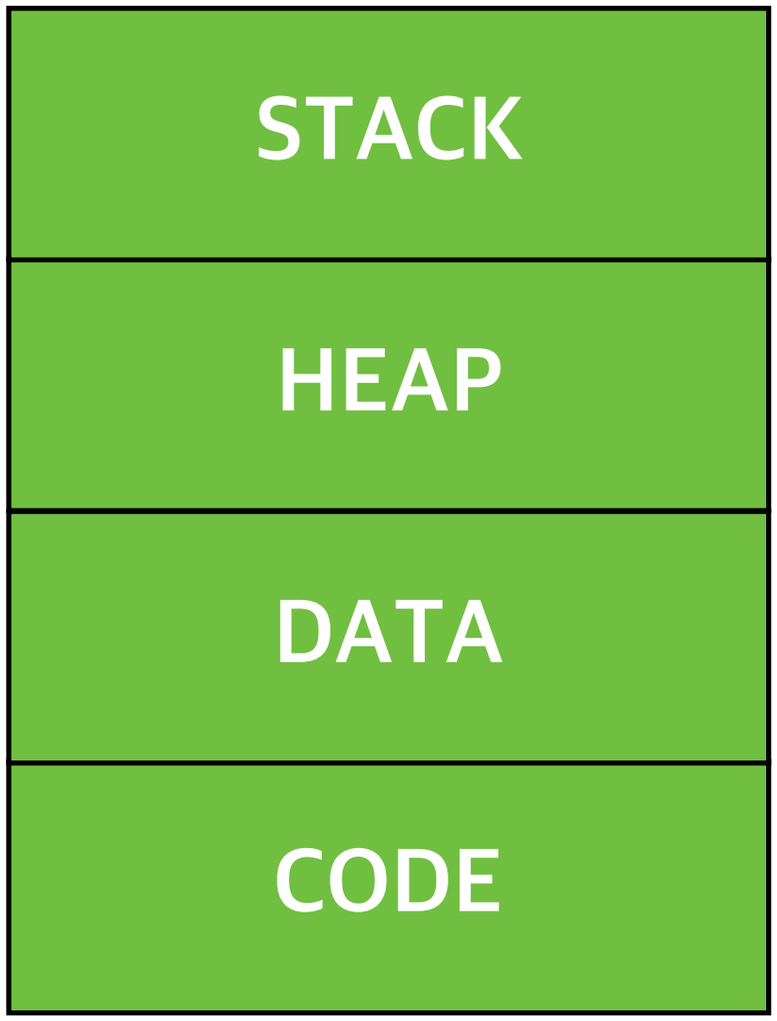
```
static let number:Int = 5
```

Memory구조 파악하기

다음 코드가 메모리에 어떻게 들어 갈까요?

```
func sumTwoNumber(num1:Int, num2:Int) -> Int
{
    return num1 + num2
}
```

Memory구조 파악하기



num1 = 3, num2 = 4

```
sumTooNumber(num1:3, num2:4)
```

프로그램 code 저장

```
func sumTwoNumber(num1:Int, num2:Int)
-> Int {
      return num1 + num2
}
```

Fast campus

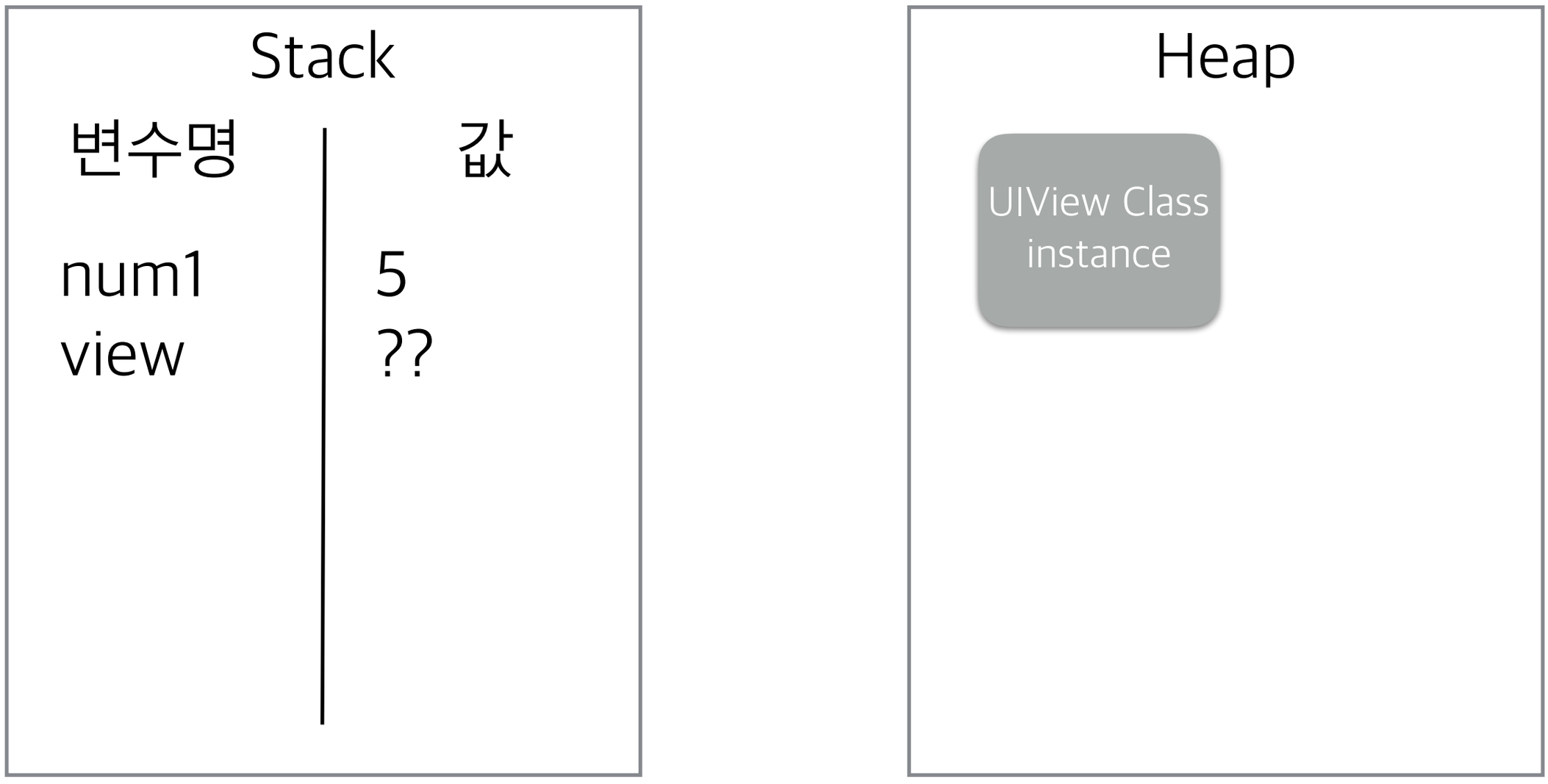
두 코드의 차이점

```
let num2:Int = 5
```

```
let lb:UIView = UIView()
```

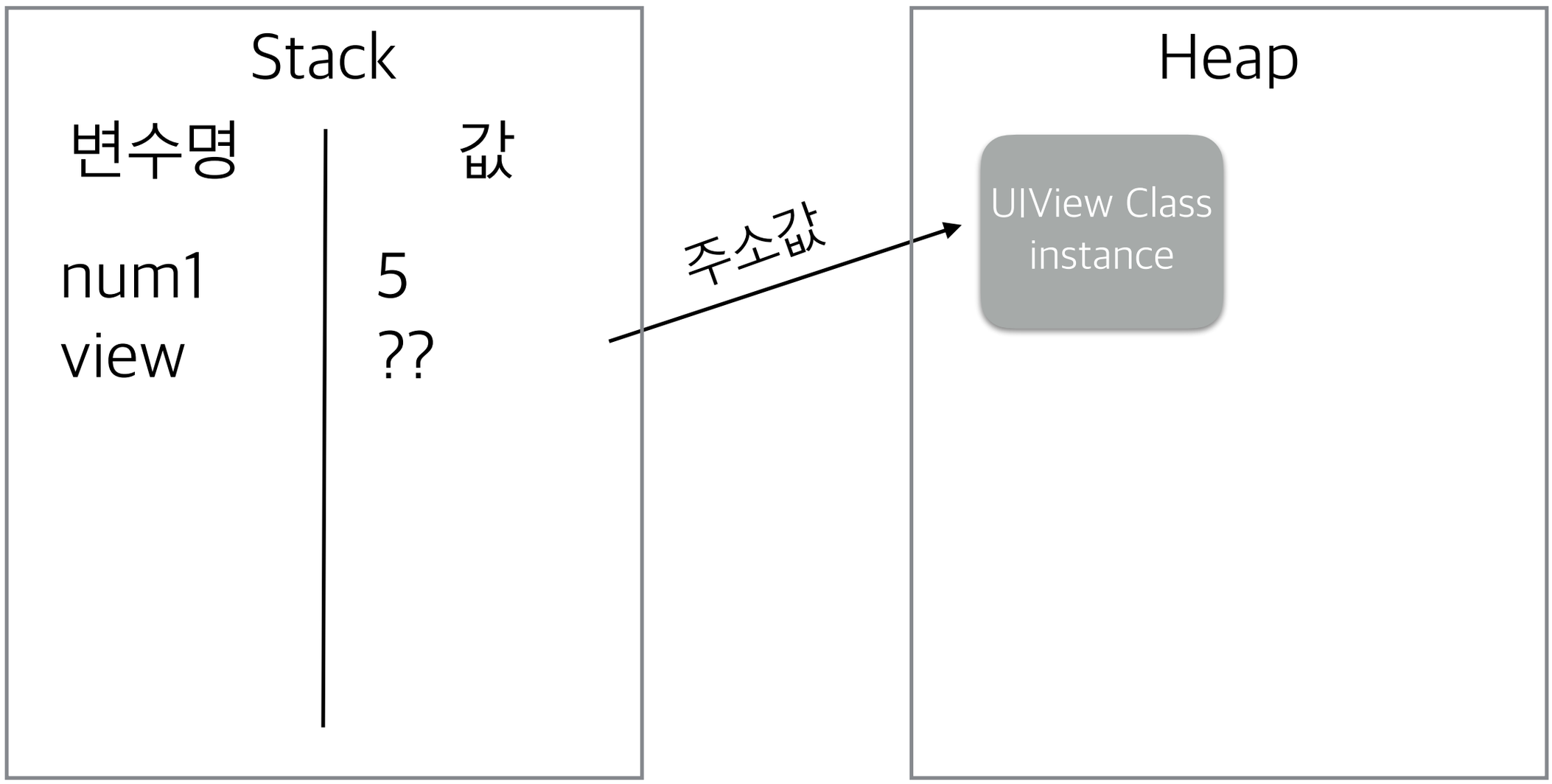
두 코드의 차이점

```
let num1:Int = 5
let view:UIView = UIView()
```



두 코드의 차이점

```
let num1:Int = 5
let view:UIView = UIView()
```



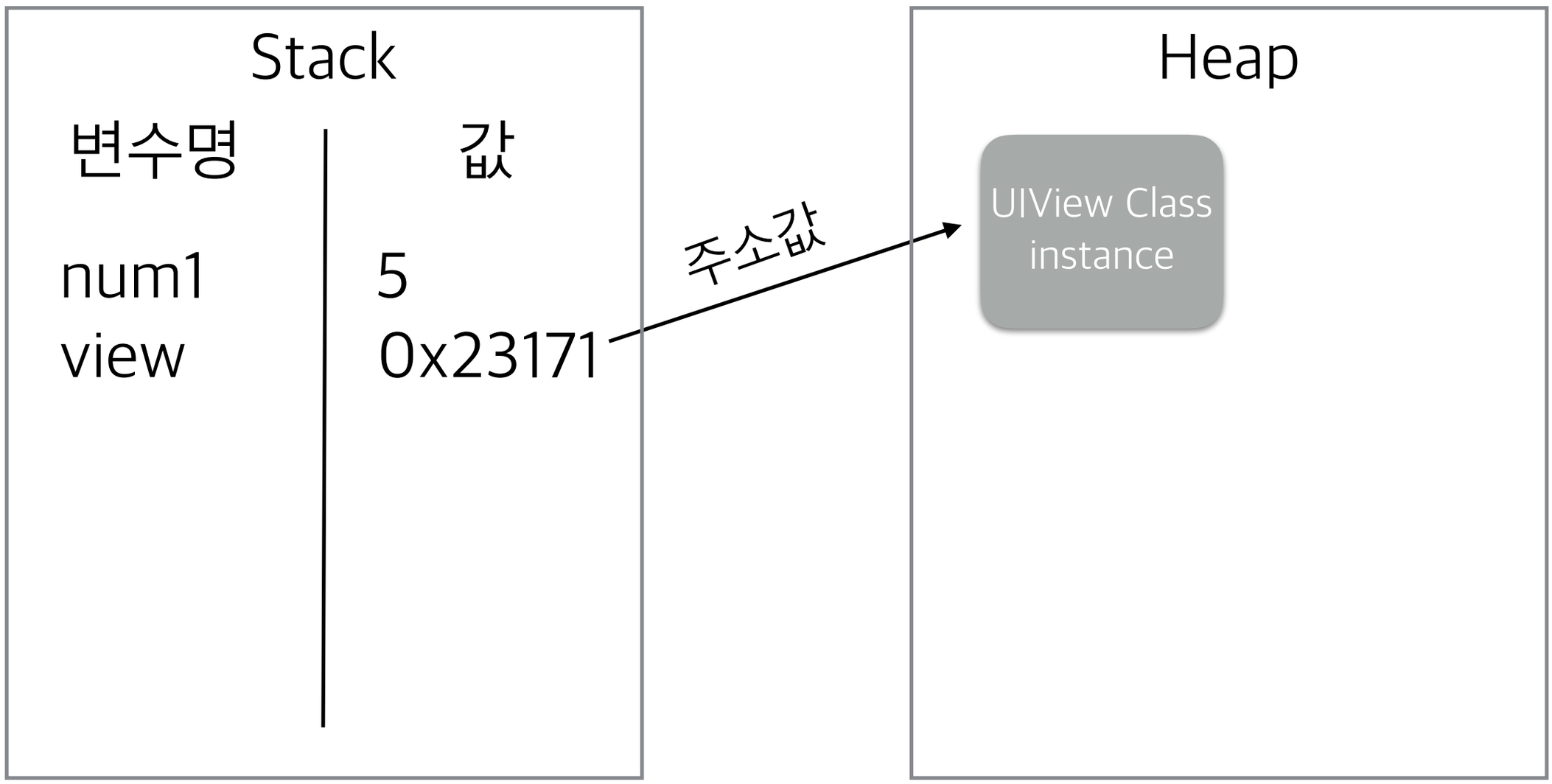
Pointer

- 포인터(pointer)는 프로그래밍 언어에서 다른 변수, 혹은 그 변수의 메모리 공간주소를 가리키는 변수를 말한다.

-Wikipedia-

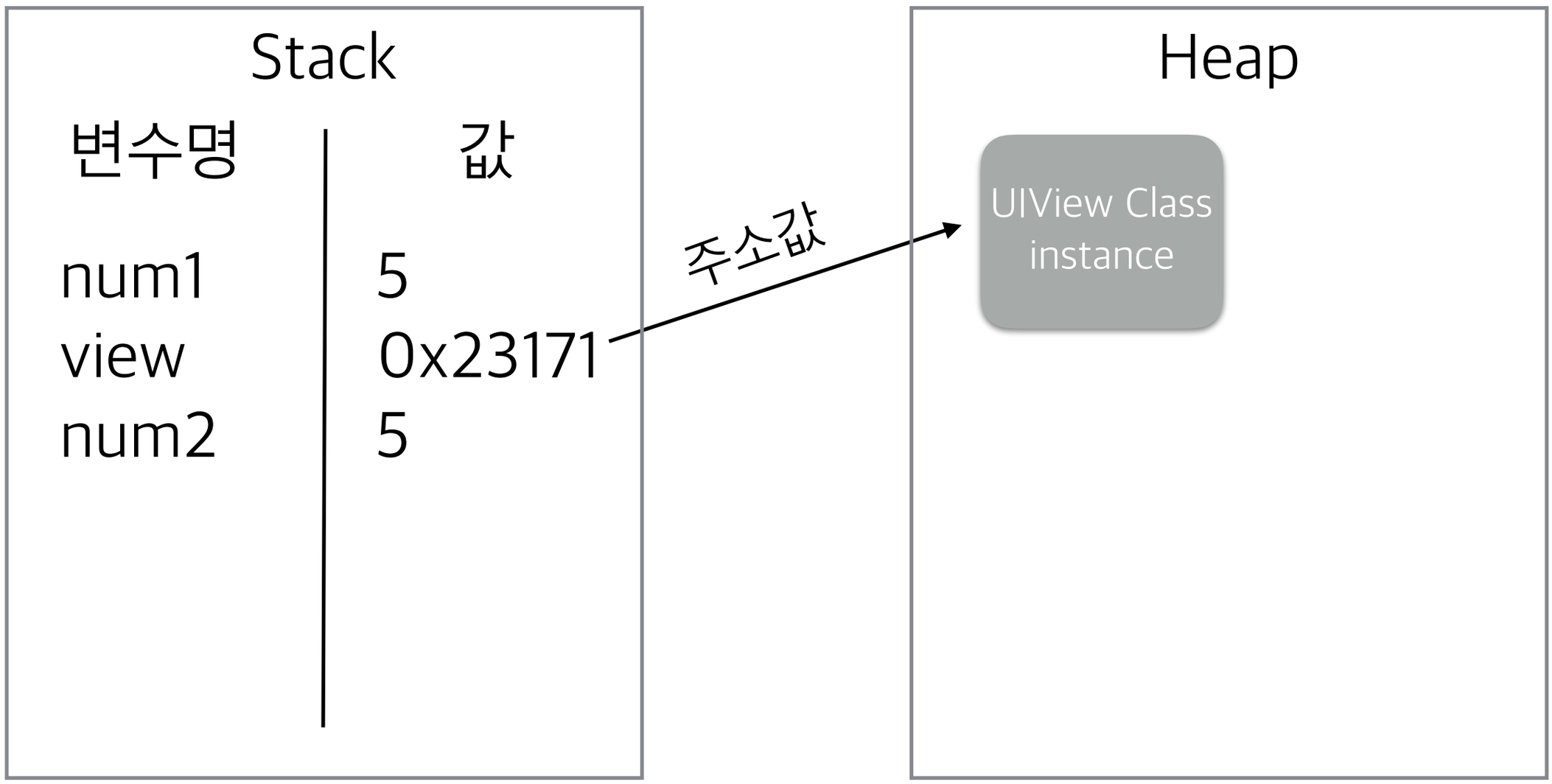
Pointer

```
let num1:Int = 5
let view:UIView = UIView()
```



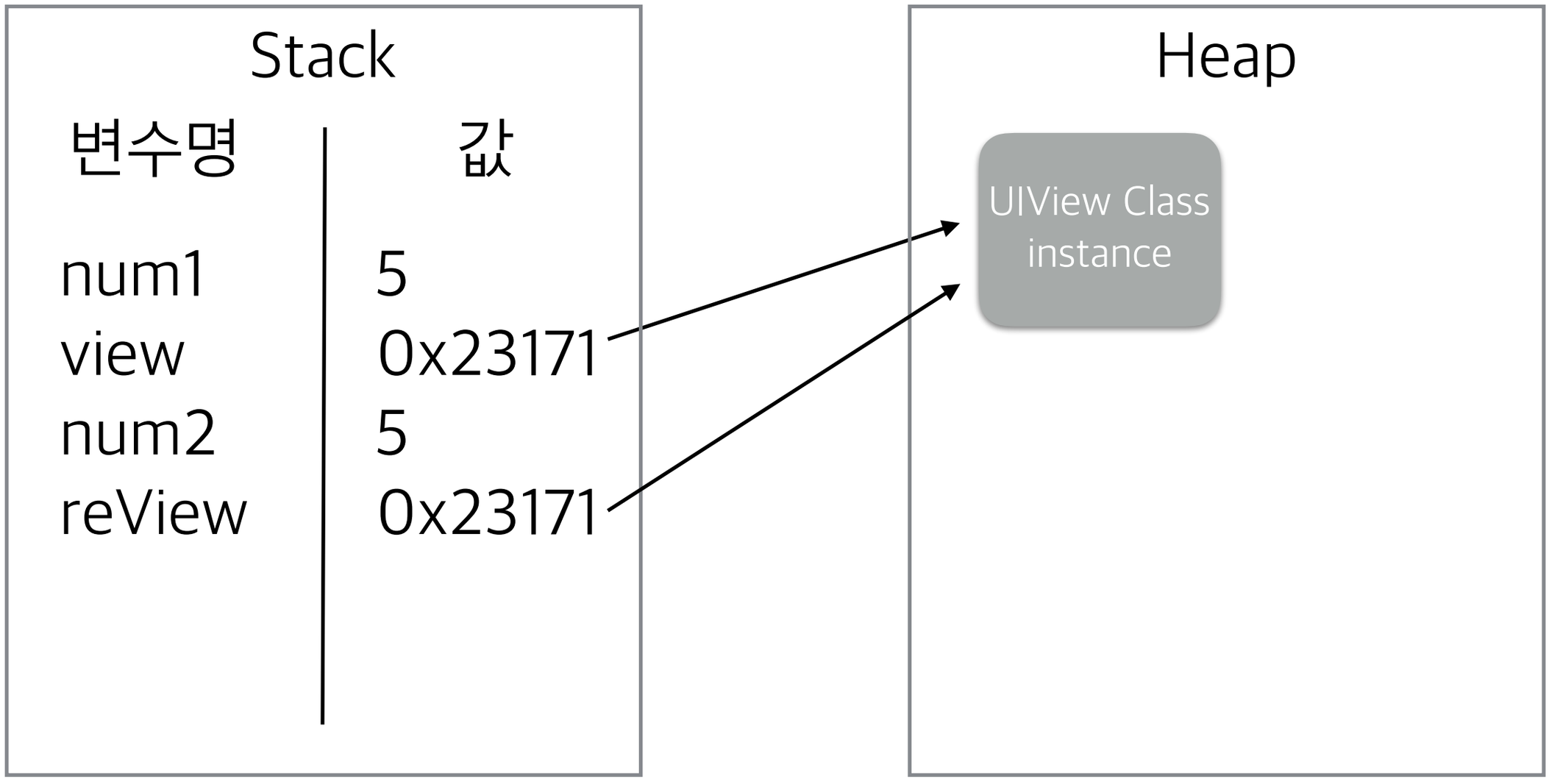
Pointer

```
let num1:Int = 5
let view:UIView = UIView()
let num2 = num1
```



Pointer

```
let view:UIView = UIView()
let num2 = num1
let reView:UIView = view
```



결과값은?

```
let view:UIView = UIView()
let reView:UIView = view
view.backgroundColor = .red
//현재 view의 배경색은? reView의 배경색은?

reView.backgroundColor = .gray
//현재 view의 배경색은? reView의 배경색은?
```

Classes VS Structures

- Class는 참조 타입이며, Structure는 값 타입이다.
- Class는 상속을 통해 부모클래스의 특성을 상속받을수 있다.
- Class는 Type Casting을 사용할수 있다.(Structure 불가)
- Structure의 프로퍼티는 instance가 var를 통해서 만들어야 수정 가능하다.
- Class는 Reference Counting을 통해 인스턴스의 해제를 계산합니다.
- Calss는 deinitializer를 사용할수 있습니다.

Deinit

```
class Student {

    init() {
        //인스턴스 생성시 필요한 내용 구현
    }

    deinit {
        //종료직전 필요한 내용 구현
    }


}
```

Classes VS Structures

- 어떤걸 선택해서 써야할까요?
- 기본 SDK에 클래스와 구조체의 예제를 찾아봅시다.

애플 가이드라인

- 연관된 간단한 값의 집합을 캡슐화하는 것만이 목적일 때
- 캡슐화된 값이 참조되는 것보다 복사되는 것이 합당할 때
- 구조체에 저장된 프로퍼티가 값타입이며 참조되는 것보다는 복사되는 것이 합당할때
- 다른 타입으로부터 상속받거나 자신이 상속될 필요가 없을 때